REMARKS

Claims 1 and 3-18 remain pending in the patent application.

The Indefiniteness Rejection

Claims 19 and 20 are rejected under 35 USC section 112, first paragraph as not being disclosed in the specification.

Claims 19-20 are canceled.

The Prior Art Rejection

The main independent claims are rejected as being anticipated by Perrin et al. (US 2004/0088153).

The main claims are amended to recite that the specific code character comprises an identifier field having information to represent an identifier which is reserved for private use and that is not any character specified in character set standards, a character field having information to represent the illegal character, and a position field having information to represent the position of the illegal character in the filename, as best shown in Figure 2.

To the extent that the rejection might be applied to the main independent claims, as amended, it is respectfully traversed because Perrin et al. does not teach or suggest a file name encoding/decoding technique featuring replacing an existing character with a specific code character having information coded therein about the illegal character itself, where the specific code character comprises:

an identifier field having information to represent an identifier which is reserved for private use and that is not any character specified in character set standards,

a character field having information to represent the illegal character, and

a position field having information to represent the position of the illegal character in the filename, as recited in the main independent claims, as recited in the main independent claims .

The claimed invention provides a simple symmetrical encoding for filenames to replace each illegal character with a respective specific code character.

Typically, filenames use 16 bit Unicode characters so there is 16 bits to encode one illegal character. According to the present invention, the encoder represents the position of the illegal character in the filename with 8 bits (2^8 = 256) and maps the illegal character itself using four bits that represent the Unicode character, as described in the patent application on page 3, lines 10-16.

In operation, the encoder identifies the specific code character from other characters by using the four most significant bits (MSB) as an illegal character indicator, e.g. using the Unicode designation "1110 = E". In the Unicode standard, code areas from E000 to F8FF are reserved for private or internal business use, and available for use as such an indicator.

In one embodiment, the specific code character can be placed at the end of the filename, before the commonly used filename extension. For example, this would make an illegal filename "ale?x.jpg" look like "alex .jpg", where the blank space represents the coded character.

The claimed invention makes it possible to receive files, containing illegal characters, from an external source connected to the mobile terminal through WAP, UNIX Server or McIntosh and store the files into the file system without corrupting the filenames. The solution also makes it possible to decode the filenames back to the original format, recreating the illegal characters.

The claimed invention also makes it possible for the mobile terminal to be used as a mass storage device for popular Apple McIntosh operating system even though the file system in mobile terminal does not directly support the MacOS operating system.

In effect, the claimed invention relates to and involves an encoding modification technique where 16 bits (or 32/48/64/128 bits etc), which normally are used to store character mapping information, could be utilized in a different way as shown in Figure 2 of the patent application. Since there is only a restricted set of illegal characters, only four bits are needed to mark them (i.e. the claimed invention uses a customized character mapping) as illustrated in Figure 2a and 2b. In the example shown in Figures 2a, 2b, the four most significant bits (MSBs) are used as "illegal character indicator" when set in a given way e.g. ´1110´ and thus a filesystem knows that all characters within a filename that have "illegal character indicator" MSBs are treated in a special way (i.e. not as normal Unicode characters but as characters informing about illegal characters that have been removed from the filename).

Further, all those characters with "illegal character indicator" are shown as blank spaces by the filesystem (by contrast, in standard Unicode a blank space is 0x0020 but in the claimed invention those illegal characters coded as 0xE... are also shown as blank spaces). Consequently, filenames like "SunEarth .txt", "Sun>Earth.txt", "Sun?Earth.txt", "Sun:Earth.txt" all would look the same "SunEarth .txt" (if multiple illegal characters then there is more blank spaces at the end of the filename, e.g. "Sun<>Earth?.txt" is shown as "SunEarth .txt"). So a user doesn't see any difference between those filenames and hence there may be several files with similar looking names in the same directory but in the bit level there are differences.

As an example, the illegal character '>' in "Sun>Earth.txt" would be coded as '1110011100000011' (0xE703) and shown as a blank space at the end of the filename since 7 is used for '>' (see Figure 2b), and 3 is the position of the character since the numbering starts from 0 (i.e. fourth character in the filename has position 3, where with 8 bits one can locate illegal characters within filenames having length of 256 characters). If there were more illegal characters those would be coded with a similar syntax and placed at the end of filename in the order of appearance.

In the case of the filename "ale?x.jpg" which would be shown as "alex .jpg" in a filesystem not allowing '?' in filenames, those allowed characters 'a', 'l', 'e' and 'x' are coded in a standard way whereas '?' is removed and a special

character having the "illegal character indicator" (e.g. 0xE) and shown as a blank space is coded as 0xE403 to indicate that it should be decoded as a '?' (see conversion table of Figure 2b) and placed as fourth character from the beginning of the filename (index 3) when transferred to a filesystem allowing such characters.

Moreover, the aforementioned discussion concentrates on a scheme where illegal characters are shown as blank spaces at the end of a filename. However, the scope of the invention is not intended to be limited to the same. For example, one may replace an illegal character with a specific code character that has 'illegal character indicator' bits (4 MSBs) set in a given way (e.g. 1110). Thus, in such a situation "ale?x.jpg" would be shown as "ale x.jpg" and the character shown as blank space would be coded as 0xE403 in view of Figure 2.

In contrast to the claimed invention, in Perrin et al. an emulation library converts a requested file name to a valid Win32 filename (e.g., by replacing invalid Win32 characters with valid Win32 characters), as described in the second sentence of paragraph 45. Moreover, when returning information related to this file, the emulation library will convert the filename back to the original filename, as described in the third sentence of paragraph 45.

For example, in Perrin et al. if an illegal character like "/" is detected in a filename like "paper/", and replaced with a legal replacement character like --1-- forming a replacement filename --paper1--, the legal replacement character --1-- is not a specific code character having information coded therein about the illegal character itself, as claimed. Instead, a person skilled in the art would appreciate that the emulation library itself maintains the needed information to convert the character from the original illegal character "/" to the legal replacement character --1--, and vice

versa to re-convert the legal replacement character --1-- back to the original legal character "/". The legal replacement character --1-- does not have information coded therein about the illegal character itself, as claimed.

In response to the reasoning on page 2 of the office action, it is respectfully submitted that when the filename "paper/" is changed to a replacement filename --paper1--, then the specific code character --1-- (i.e. replacement character) for the illegal character (i.e. "/") clearly does not have information coded therein about the illegal character itself (i.e. "/"). Instead, as a person skilled in the art would appreciate that the emulation library itself has and maintains the needed information to convert the requested file name to a valid Win32 filename (e.g., by replacing invalid Win32 characters with valid Win32 characters), and to re-convert the valid Win32 filename back to the original invalid Win32 filename.

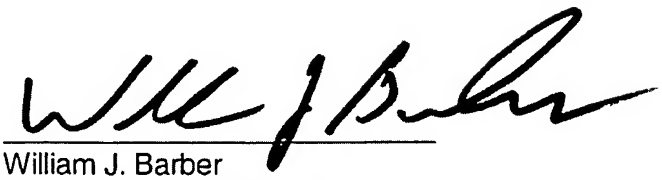
Remaining Dependent Claims

The remaining dependent claims depend directly or indirectly from one or more of the aforementioned independent claims, contain all the limitations thereof, and are deemed patentable for all the reasons set forth above.

Conclusion

For all these reasons, reconsideration and early allowance is respectfully requested.

Respectfully submitted,



William J. Barber
Attorney for the Applicant
Registration No. 32,720

27 February 2008

WARE, FRESSOLA, VAN DER SLUYS
& ADOLPHSON LLP
Customer No. 004955
Bradford Green, Building Five
755 Main Street, P.O. Box 224
Monroe, CT 06468
(203) 261-1234